Declarative Language for Geometric Pattern Matching in VLSI Process Rule Modeling

2019 International Symposium on Physical Design
San Francisco, CA

Gyuszi Suto, Geoff S. Greenleaf, Phanindra Bhagavatula, Heinrich R. Fischer
Sanjay K. Soni, Brian H. Miller, Renato F. Hentschke

Intel Corporation
Regular Expressions

\[ [Aa][ie]ro?plane \quad \ll{\text{pattern of interest}} \]

Airplane
aeroplane
aerooplane

Which one of these words will match the regular expression?
Regular Expressions

\[ [Aa][ie]ro?plane \] \< pattern of interest \]

Airplane \< match \np aeroplane \< match \np aerooplane \< no match

\text{assumed spatial adjacency}
Regular Expressions

([Aa][ie]ro?plane ){2}  << pattern of interest

I'm just a aeroplane poor boy and nobody loves me, He's just a poor boy from
a poor family, Spare him his life from this monstrosity, Easy come easy go
will you let me go, Bismillah, Airplane aeroplane no we will aiirplane not
let you go, let him go, Bismillah, we will not let you go, Airplane let him
go, Bismillah, we will not airpplan let you go, let me go, (Will not let you
go) let me go (never, never let you go) let me Airplane Aeroplane go (never
let me go), Oh oh no, no, no, no, no, no, Oh mama mia, mama mia, mama
mia let me go, Beelzebub has a devil put aside for me for me for me for me
aeroplane Airplane for me airplane for you Airplane for them aeroplane.
Regular Expressions

([Aa][ie]ro?plane ){2}  << pattern of interest

I’m just a aeroplane poor boy and nobody loves me, He's just a poor boy from a poor family, Spare him his life from this monstrosity, Easy come easy go will you let me go, Bismillah, Airplane aeroplane no we will aiirplane not let you go, let him go, Bismillah, we will not let you go, Airplane let him go, Bismillah, we will not airpplan let you go, let me go, (Will not let you go) let me go (never, never let you go) let me Airplane Aeroplane go (never let me go), Oh oh no, no, no, no, no, no, Oh mama mia, mama mia, mama mia let me go, Beelzebub has a devil put aside for me for me for me aeroiplane Airplane for me airplane for you Airplane for them aeroplane.

• Formal
• Non-ambiguous
• Expressive
• Compact
• Can be typed in by non-programmers
• Wouldn’t it be nice to have to a similar language for VLSI layout patterns?
Simple Example

Type Fuselage Wire layer=metal2 orient=H dy=10

Pattern Airplane
AND
  f Fuselage ; there exists a fuselage

match
match
match
match
match
match
match
Simple Example

Type Fuselage Wire layer=metal2 orient=H dy=10

Pattern Airplane

AND

\[ f \text{ Fuselage} | f.\text{delta}(H) \text{ in } \{70, [100, 120], 140\} \]

match

match

match

match

match

no match, does not satisfy the \( \text{delta}(H) \) constraint

90

match

guard expression such that horizontal dimension is in the given interval set
Simple Example

**Type** Wing  **Wire** layer=metal1 orient=V

**Pattern** Airplane

**AND**
- \( f \) Fuselage \( | f . \text{delta}(H) \) in \{70, [100, 120], 140\}
- \( w \) Wing \( | w . \text{delta}(V) \) in \{70, 110\}
  - \( w . \text{yc} \) == \( f . \text{yc} \)
  - \( f . \text{contains}(w, H) \)

...
Simple Example

Type Cockpit Via layer=via2

Pattern Airplane

AND

\[ f \text{ Fuselage} \mid f.\delta(H) \in \{70, [100, 120], 140\} \]
\[ w \text{ Wing} \mid w.\delta(V) \in [70, 110] \]
\[ w.\text{yc} = f.\text{yc} \]
\[ f.\text{contains}(w, H) \]
\[ p \text{ Cockpit} \mid f.\text{contains}(p) \]
\[ \text{NOT} \ w.\text{intersects}(p) \]
Example of a Complete Airplane

Type Tail Wire layer=metal3 orient=V

Pattern Airplane

AND

f Fuselage | f.delta(H) in {70, [100, 120], 140}
w Wing | w.delta(V) in [70, 110]
  w.yc == f.yc
  f.contains(w, H)
p Cockpit | f.contains(p)
  NOT w.intersects(p)
t Tail | t.yc == f.yc
  f.contains(t, H)
Wing With Engines Pattern

Type Engine  Wire layer=metal4 orient=H

Pattern WingWithEngines
AND
  w Wing
  e1, e2 Engine  |  w.contains(this, V)
         this.contains(w, H)
  e1.generalized_intersection(e2).yc == w yc
Pattern Glider
AND
a Airplane
; there does not exist a WingWithEngines
; such-that its wing is identical
; with a’s wing
NOT w WingWithEngines | w.w == a.w
Jet Pattern

**Pattern** Jet

**AND**

a Airplane

; there does exist a WingWithEngines

; such that its wing is identical

; with a’s wing

w WingWithEngines | w.w == a.w
Jet Pattern, with Windows

Type Window Via layer=via1

Pattern Jet
AND
a Airplane
; there does exist a WingWithEngines
; such that its wing is identical
; with a’s wing
w WingWithEngines | w.w == a.w
let windows = Set {Window} | a.f.contains(this)
windows.count in [3, 5]
Jet Pattern, with Windows on Grid

Grid g offset=0 period=10 orient=V

Pattern Jet
AND
  a Airplane
  ; there does exist a WingWithEngines
  ; such that its wing is identical
  ; with a’s wing
  w WingWithEngines | w.w == a.w
let windows = Set {Window} | a.f.contains(this)  
  this.xc on g
windows.count in [3, 5]
Formation Pattern

Pattern Formation
AND
a1 Jet
a2 \{Jet, Glider\} ; a2 is Jet or Glider
a1.a.p.euclidean_distance(a2.a.p) < 60
Rule CloseFormations
AND
f1, f2 Formation
f1.bbox.intersects(f2.bbox)

; there does not exist an airplane a
NOT a Airplane |
OR
a.bbox.spacing(f1.bbox,V)< 20 false
a.bbox.spacing(f2.bbox,V)< 20 false
CloseFormations Pattern

**Rule CloseFormations**

**AND**

- f1, f2 Formation
- f1.bbox.intersects(f2.bbox)

; there does not exist an airplane a

**NOT**

| a Airplane |

**OR**

| a.bbox.spacing(f1.bbox,V) < 20 | true |
| a.bbox.spacing(f2.bbox,V) < 20 | false |

true
false
true
false
false

no match
Automatic Test Layout Generation

**Pattern** Formation
AND
a1 Jet
a2 Glider
\( a1.a.p.euclidean\_distance(a2.a.p) < 60 \)

**Rule** CloseFormationsSimplified
AND
f1, f2 Formation
f1.bbox.spacing(f2.bbox, V) in [150, 300]

Test Layout Automatically Generated from the rule above
More Auto-generated Test Layouts
All these test layouts were automatically generated from the rule `CloseFormationsSimplified`, using mathematical solvers. The number of tests to be generated is an input parameter, can be in the thousands or higher.
Why?

Rule Expert

- Not formal
- Not complete

Constraints
Rules
Written
In English

- Formal
- Complete
- Imperative
- Not human consumable
- May not encode original intent

Runset Code

- Ambiguous
- Labor intensive
- Error-prone

PD Tool
Tech Model

SAT Solver

Test Layout
Future, with the Declarative Language

**Rule Expert**

**Rule** `CloseFormationsSimplified`

**AND**

- `f1`, `f2` Formation
- `f1.bbox.intersects(f2.bbox)`

**Formal, Complete, Declarative**

**Formal**

- Complete
- Imperative
- Encodes original intent

**Runset Code**

**Automated**

**Online Checker**

**Test Layout**

**PD Tool**

**Tech Model**

**SAT Solver**

**Unambiguous Automated Rigurous**
What

• A declarative language (think math, expressions, no state-change)
• Formal (computer readable)
• Compact, expressive
• Easy to teach, easy to write, easy to understand
• Support rules of any complexity, w/o need to modify language
• Aims to be the de-facto language for rule communication
  • Human to human, human to machine, machine to machine
• Expansive geometric function set
• Polygon Set support
• Advanced features: Patterns, Nested Patterns, Exceptions, Custom Functions, Grids, Sets, etc.
Status. Future Work.

• Language productized
• Online checker productized
• Automatic Test Layout Generator in Alpha release
• Automatic conversion to PD tool model – proof of concept
• Language open sourced to Si2 under OPAL name, older version
BACKUP
Actual layout used in unit testing patterns presented in this document